Määrittelydokumentti
Shakki-AI
Sebastian Falk

Minun työni aihe tulee olemaan tekoäly shakkipelille. Teen tämän tekoälyn jo valmiille shakkipelille jonka tein ohjelmoinnin harjotustyöhön. Minä tulen toteuttamaan tekoälyn käyttämällä alpha-beta karsintaa. Minä en toteuta omia tietorakenteita, minä käytän ArrayList:iä mutta sain kuulla että minun ei tarvitse toteuttaa sitä itse työn vaativuuden takia.

Ongelmani on yksinkertaisesti saada ohjelma löytämään hyviä siirtoja shakkipelissä, sopivassa ajassa. Tässä työssä aika ei kuitenkaan ole tärkein asia, alpha-betassa voi lisätä syvyyttä paljolla jos haluaa ja silloin suoritusaikakin kasvaa paljon. Valitsin alpha-beta karsinnan koska se on aika suosittu algoritmi shakki-tekoälyissä ja myös koska se on minulle aikaisemmasta tuttu.

Ohjelma saa syötteenä ruudukon nappuloista ja halutun syvyyden jolla alpha-betaa suoritetaan. Ruudukko on kaksiuloitteinen joka koostuu 64:stä ruudusta joissa on joko nappula tai ei. Ohjelma käy läpi tämän ruudukon ja lisää listaan kaikkien ruudukon nappuloitten siirrot. Jokaiselle näistä siirroista ohjelma tekee siirron ja kutsuu sitten joko alphaBetaMax (kun on valkoisen vuoro) tai alphaBetaMin (kun on mustan vuoro) metodia pelilaudalla jossa siirto on tehty. Jos arvo joka palautetaan on parempi kuin aikaisempi arvo niin parasta arvoa ja parasta siirtoa muutetaan. AlphaBetaMin ja alphaBetaMax laskevat pelilaudan arvon jos syvyys on 0 ja muuten ne tekevät saman asian kuin bestMove metodi, siis tekevät kaikki siirrot ja kutsuvat toista metodeista yhden pienemmällä syvyydellä.

Aina kun syvyys on 0 alphaBeta metodeissa niin kutsutaan evaluaatiofunktiota. Tämä funktio on ohjelman tärkein osa. Se laskee pelilaudalle arvon, hyvä pelilauta valkoiselle palauttaa korkean arvon ja hyvä mustalle palauttaa pienen arvon. Funktio katsoo kuinka monta nappulaa pelilaudalla on, jokaisella nappulalla on oma arvonsa (kuningas on arvokkain ja sotilas vähiten arvokas). Se katsoo myös missä nappulat ovat, jos nappula on hyvässä paikassa lisätään pisteitä, muuten vähennetään.

Kun ohjelma on lopulta mennyt kaikki siirrot läpi halutulla syvyydellä niin meillä on paras siirto alkuperäiselle pelilaudalle. Tämä siirto palautetaan ja käyttöjärjestelmä suorittaa siirron ja antaa vuoron toiselle pelaajalle.

Ohjelmalleni on hieman vaikeaa tehdä O-analyysiä. Nopeus riippuu sekä syvyydestä jolla karsintaa suoritetaan että kaikkien mahdollisten siirtojen määrästä. Eli lopullinen aikavaativuus tulee olemaan luokkaa O(n^m) jossa n on siirtojen määrä ja m on syvyys jolla ohjelmaa suoritetaan. Niinkuin sanoin, tärkein asia ei ole tässä ohjelmassa että suoritus on vain nopea mutta että siirto on hyvä.

Lähteitä:

-http://chessprogramming.wikispaces.com/Alpha-Beta
-https://chessprogramming.wikispaces.com/Evaluation